Day 6: Nonlinear models and tree-based methods

ME414: Introduction to Data Science and Big Data Analytics

LSE Methods Summer Programme

21 August 2017

Day 6 Outline

Moving Beyond Linearity

- Polynomial Regression
- Step Functions
- Splines
- Local Regression
- Generalized Additive Models

Tree-based Methods

- Bagging
- Boosting

Application example

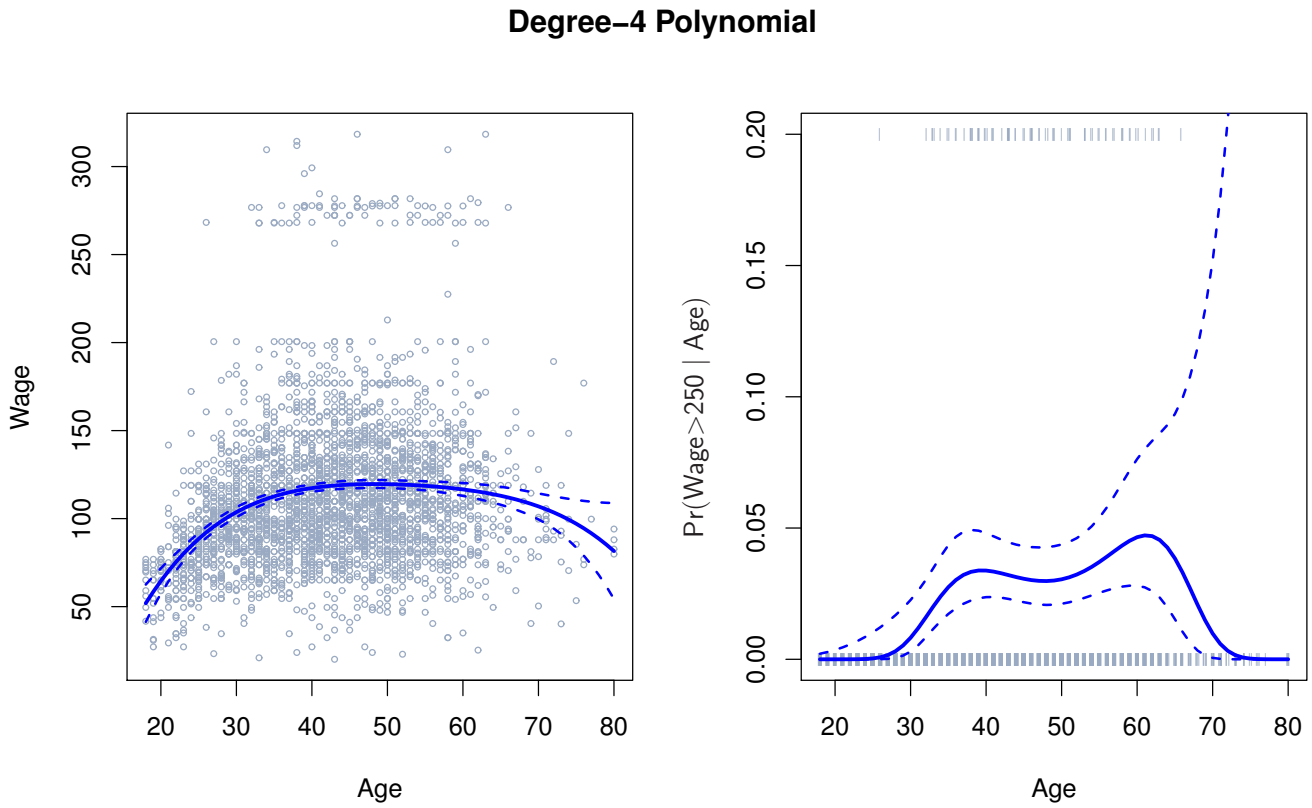
Moving Beyond Linearity

Linearity and social reality

- Social world is almost never linear.
- Often the linearity assumption is good enough.
- When linearity doesn't hold we can use
  - polynomials,
  - step functions,
  - splines,
  - local regression, and
  - generalized additive models
- These models offer a lot of flexibility, without losing the ease and interpretability of linear models.

Polynomial regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \epsilon_i$$



Details

- Create new variables $X_1 = X$, $X_2 = X^2$, etc and then treat as multiple linear regression.
- Not really interested in the coefficients; more interested in the fitted function values at any value x_0:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \hat{\beta}_3 x_0^3 + \hat{\beta}_4 x_0^4.$$

- Since $\hat{f}(x_0)$ is a linear function of the $\hat{\beta}_\ell$, can get a simple expression for pointwise-variances $\mathrm{Var}[\hat{f}(x0)]$ at any value x_0.
- In the figure we have computed the fit and pointwise standard errors on a grid of values for x_0. We show $\hat{f}(x_0) \pm 2 \cdot se[\hat{f}(x_0)]$.
- We either fix the degree d at some reasonably low value, else use cross-validation to choose d.

Details continued

- Logistic regression follows naturally. For example, in figure we model

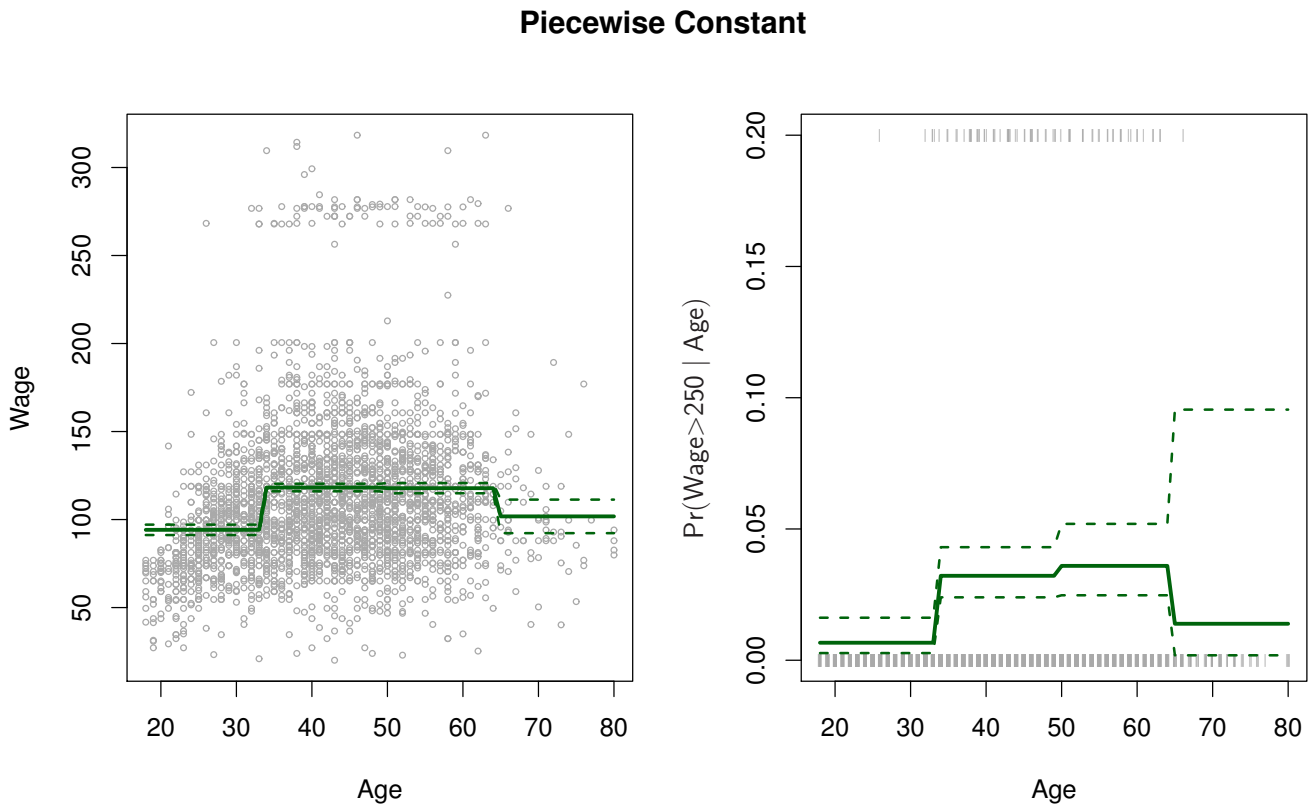
$$Pr(y_i > 250|x_i) = \frac{exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 \cdots + \beta_d x_i^d)}{1 + exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 \cdots + \beta_d x_i^d}.$$

- To get confidence intervals, compute upper and lower bounds on the logit scale, and then invert to get on probability scale.
- Can do separately on several variablesâĂŤjust stack the variables into one matrix, and separate out the pieces afterwards (see GAMs later).
- Caveat: polynomials have notorious tail behavior – very bad for extrapolation.
- Can fit using $y \sim poly(x, degree = 3)$ in formula.

Step Functions

Another way of creating transformations of a variable – cut the variable into distinct regions.

$$C_1(X) = I(X < 35), C_2(X) = I(35 \leq X < 50), \ldots, C_3(X) = I(X \geq 65)$$



Step functions continued

- Easy to work with. Creates a series of dummy variables representing each group.
- Useful way of creating interactions that are easy to interpret. For example, interaction effect of *Year* and *Age*:

$$I(\mathit{Year} < 2005) \cdot \mathit{Age}, \;\; I(\mathit{Year} \geq 2005) \cdot \mathit{Age}$$

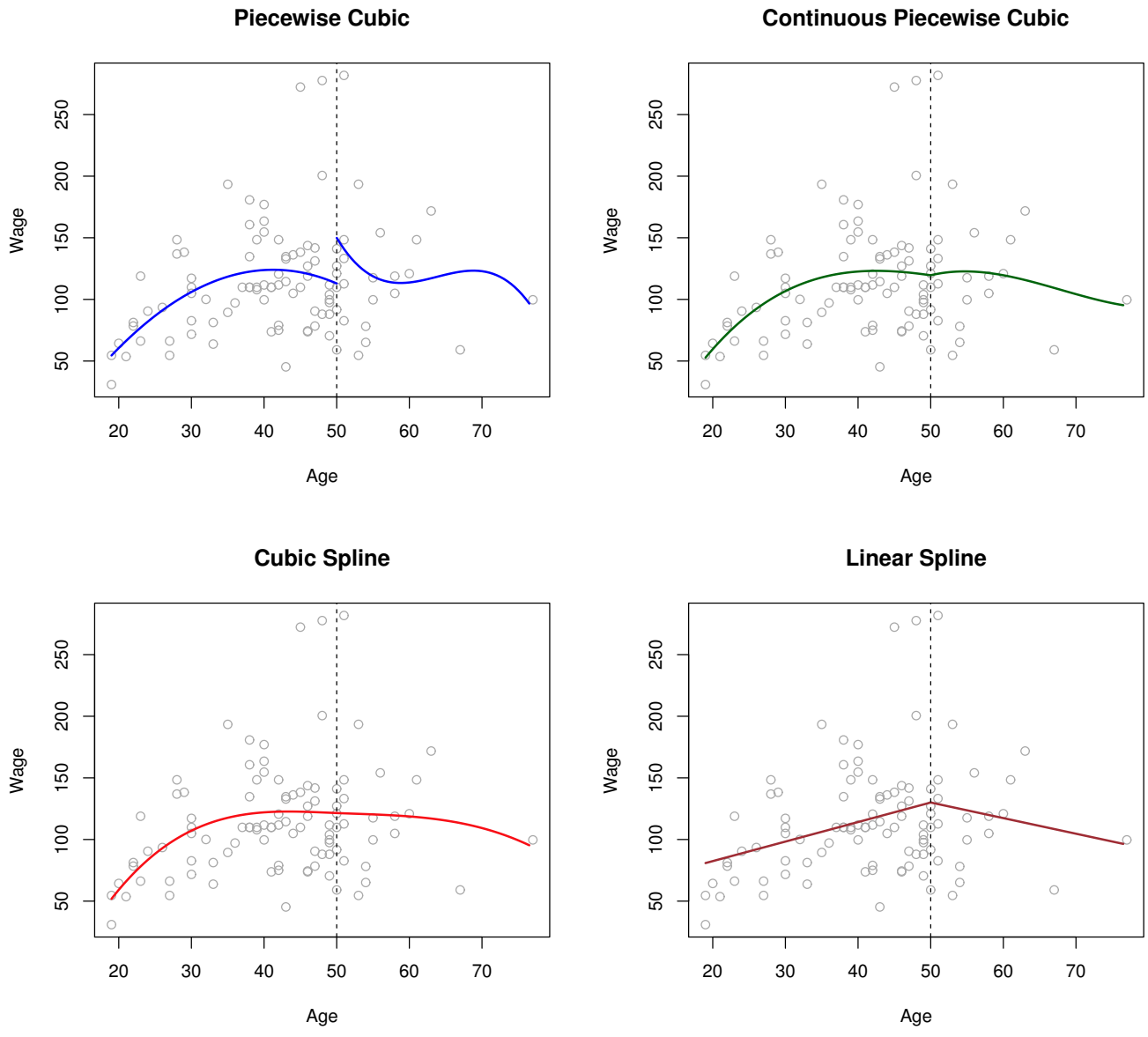
  would allow for different linear functions in each age category.
- In R: $I(year < 2005)$ or $cut(age, c(18, 25, 40, 65, 90))$.
- Choice of cutpoints or knots can be problematic. For creating nonlinearities, smoother alternatives such as splines are available.

Piecewise polynomials

- Instead of a single polynomial in X over its whole domain, we can rather use different polynomials in regions defined by knots. E.g. (see figure)

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

- Better to add constraints to the polynomials, e.g. continuity.
- Splines have the "maximum" amount of continuity.


Piecewise Cubic
Continuous Piecewise Cubic
Cubic Spline
Linear Spline
Wage
Age
50
100
150
200
250
20
30
40
50
60
70

Linear Splines

- A linear spline with knots at ξ_k, $k = 1, \ldots, K$ is a piecewise linear polynomial continuous at each knot.
- We can represent this model as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

where the b_k are basis functions.

$$b_1(x_i) = x_i$$

$$b_{k+1}(x_i) = (x_i - \xi_k)_+, \ k = 1, \ldots, K$$

Here the $()_+$ means positive part; i.e.

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k & \text{if } x_i > \xi_k; \\ 0 & \text{otherwise.} \end{cases}$$

Cubic splines

- A cubic spline with knots at ξ_k, $k = 1, \ldots, K$ is a piecewise cubic polynomial with continuous derivatives up to order 2 at each knot.
- We can represent this model with truncated power basis functions

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

where the b_k are basis functions.

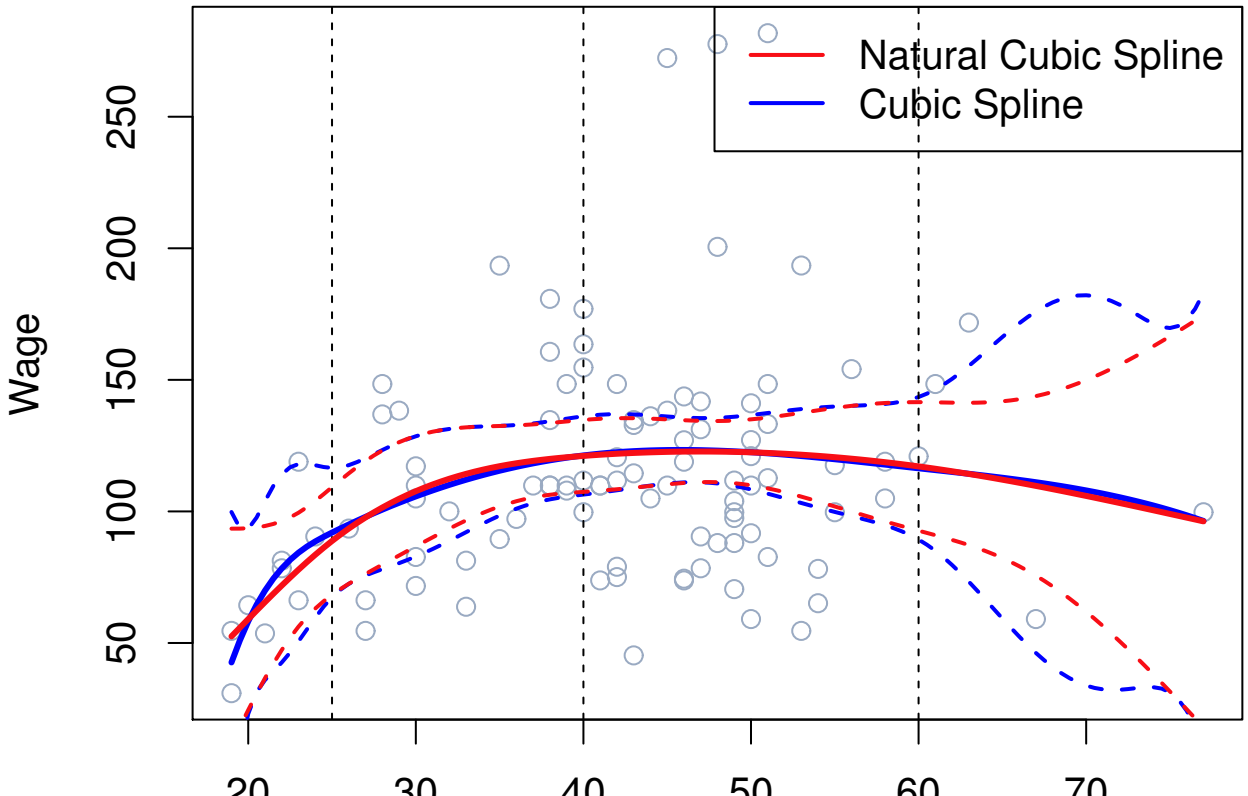
$$b_1(x_i) = x_i; b_2(x_i) = x_i^2; b_3(x_i) = x_i^3;$$
$$b_{k+3}(x_i) = (x_i - \xi_k)_+^3, \ k = 1, \ldots, K$$

where

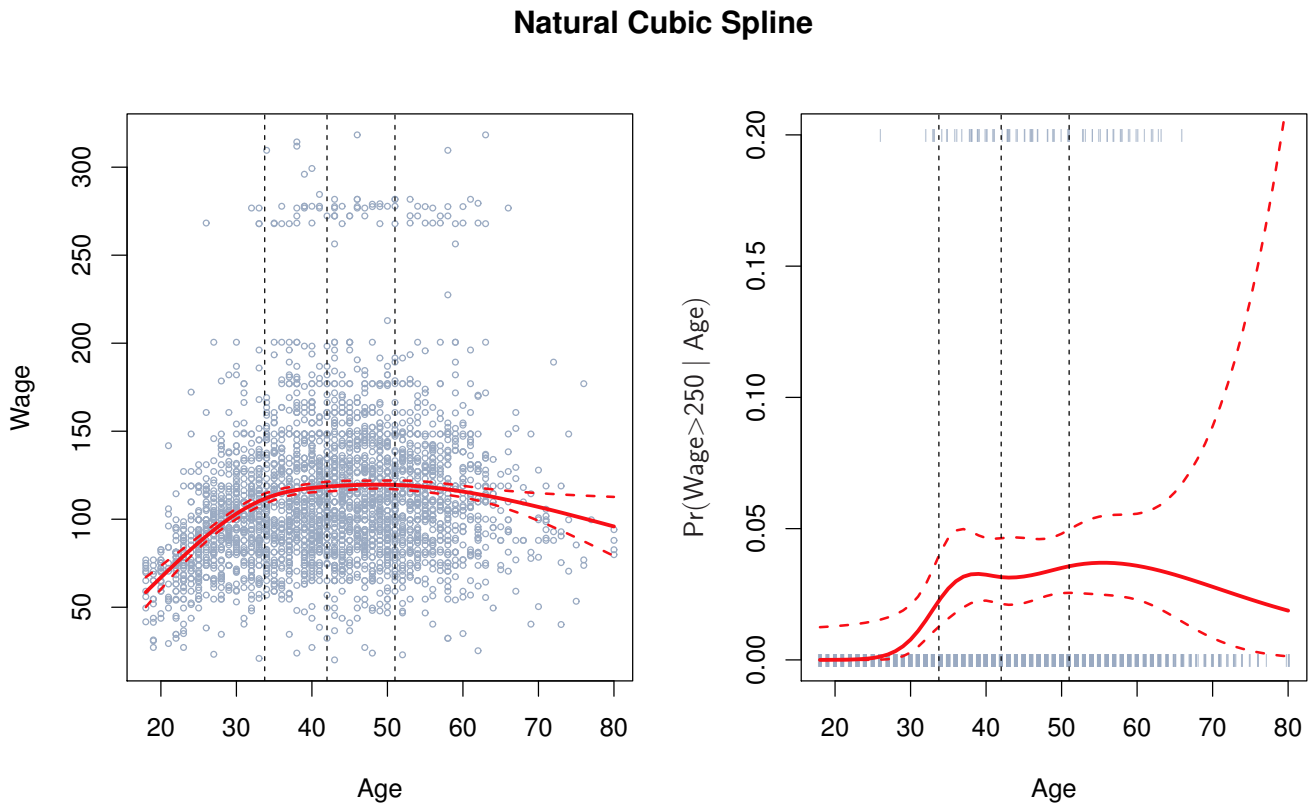
$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3 & \text{if } x_i > \xi_k; \\ 0 & \text{otherwise.} \end{cases}$$

Natural cubic splines

- A natural cubic spline extrapolates linearly beyond the boundary knots.
- This adds $4 = 2 \times 2$ extra constraints, and allows us to put more internal knots for the same degrees of freedom as a regular cubic spline.

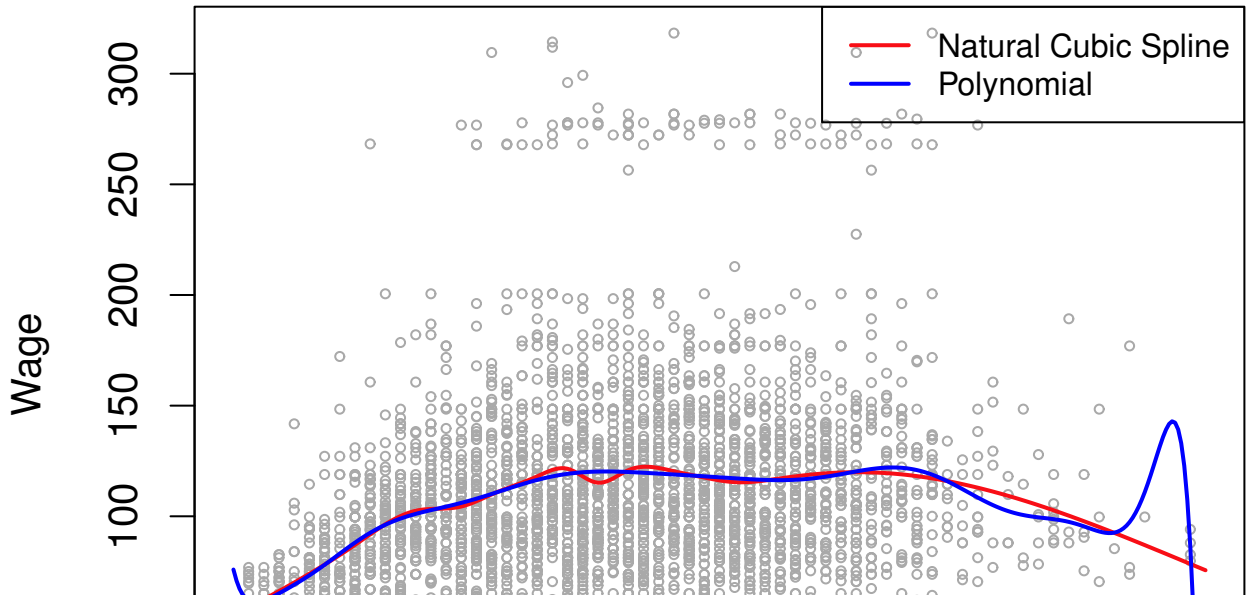


- Fitting splines in R is easy: $bs(x, \dots)$ for any degree splines, and $ns(x, \dots)$ for natural cubic splines, in package *splines*.



Knot placement

- One strategy is to decide K, the number of knots, and then place them at appropriate quantiles of the observed X.
- A cubic spline with K knots has $K + 4$ parameters or degrees of freedom.
- A natural spline with K knots has K degrees of freedom.
- On the figure, comparison of a degree-14 polynomial and a natural cubic spline, each with 15df: $ns(age, df = 14)$ and $poly(age, deg = 14)$.



Smoothing splines

- Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\underbrace{\textit{minimize}}_{g \in S} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- The first term is RSS, and tries to make $g(x)$ match the data at each x_i.
- The second term is a roughness penalty and controls how wiggly $g(x)$ is. It is modulated by the tuning parameter $\lambda \geq 0$.
- The smaller λ, the more wiggly the function, eventually interpolating y_i when $\lambda = 0$.
- As $\lambda \to \infty$, the function $g(x)$ becomes linear.

Smoothing splines continued

- The solution is a natural cubic spline, with a knot at every unique value of x_i.
- The roughness penalty still controls the roughness via λ.

Smoothing splines details

- Smoothing splines avoid the knot-selection issue, leaving a single λ to be chosen.
- The algorithmic details are too complex to describe here. In R, the function *smooth.spline()* will fit a smoothing spline.
- The vector of n fitted values can be written as $\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y}$, where $\mathbf{S}_\lambda$ is a $n \times n$ matrix (determined by the x_i and λ).
- The effective degrees of freedom are given by

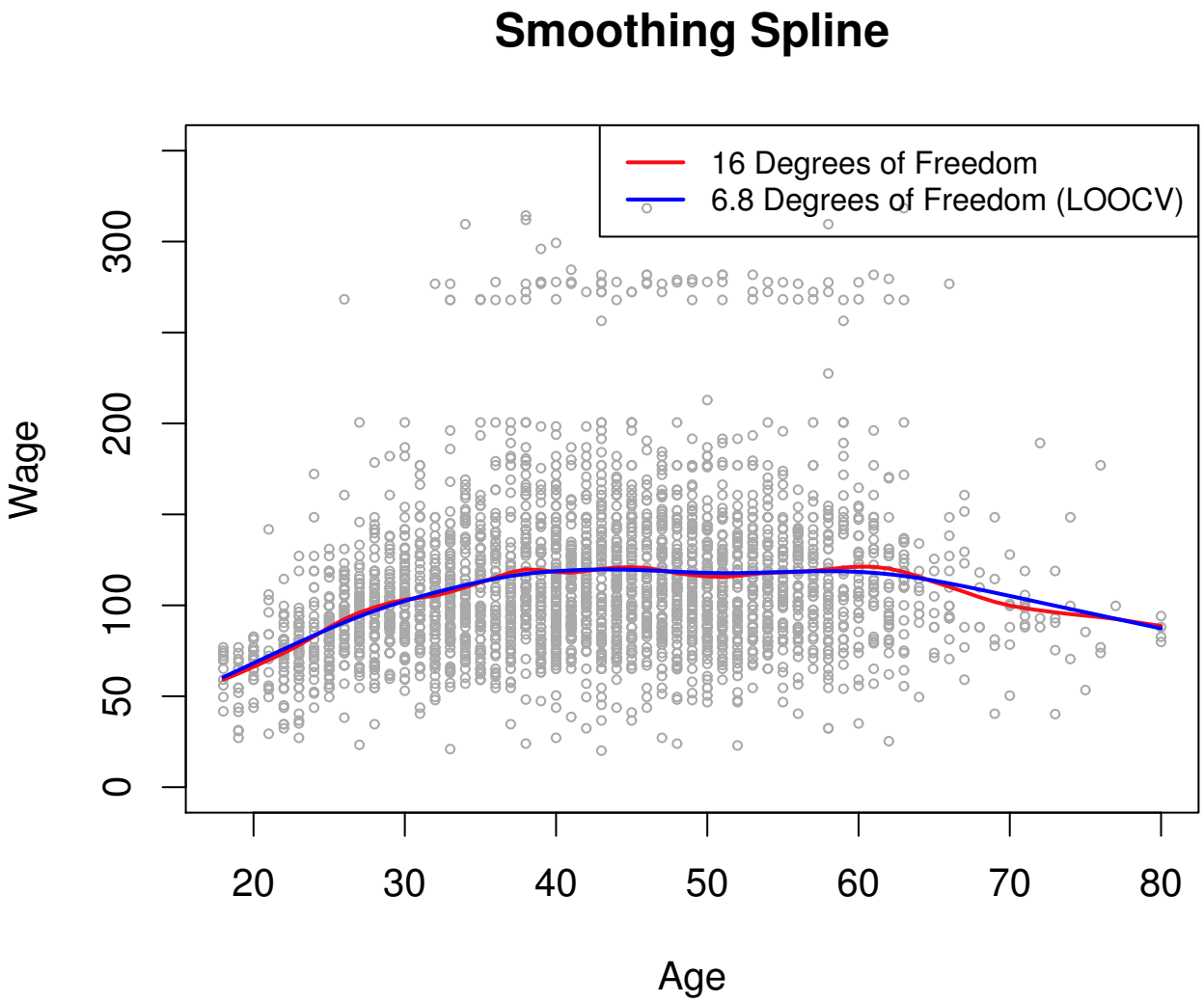
$$df_\lambda = \sum_{i=1}^{n} \{\mathbf{S}_\lambda\}_{ii}.$$

Smoothing splines – choosing λ

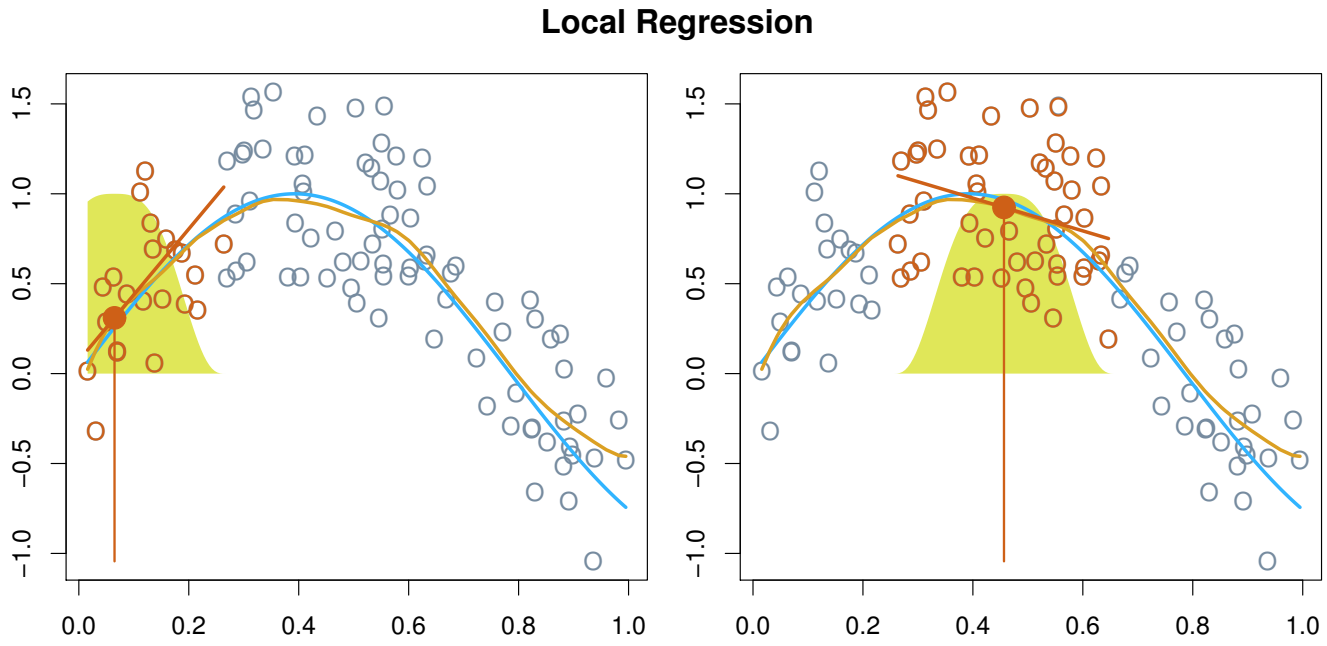
- We can specify *df* rather than λ. In R: *smooth.spline*(*age*, *wage*, *df* $= 10$).
- The LOOCV error is given by

$$\mathrm{RSS}_{cv}(\lambda) = \sum_{i=1}^{n} (y_i - \hat{g}_\lambda^{-i}(x_i))^2 = \sum_{i=1}^{n} \left[\frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}} \right].$$

  In R: *smooth.spline*(*age*, *wage*)


Smoothing Spline
16 Degrees of Freedom
6.8 Degrees of Freedom (LOOCV)
Wage
0
50
100
200
300
20
30
40
50
60
70
80
Age

Local regression

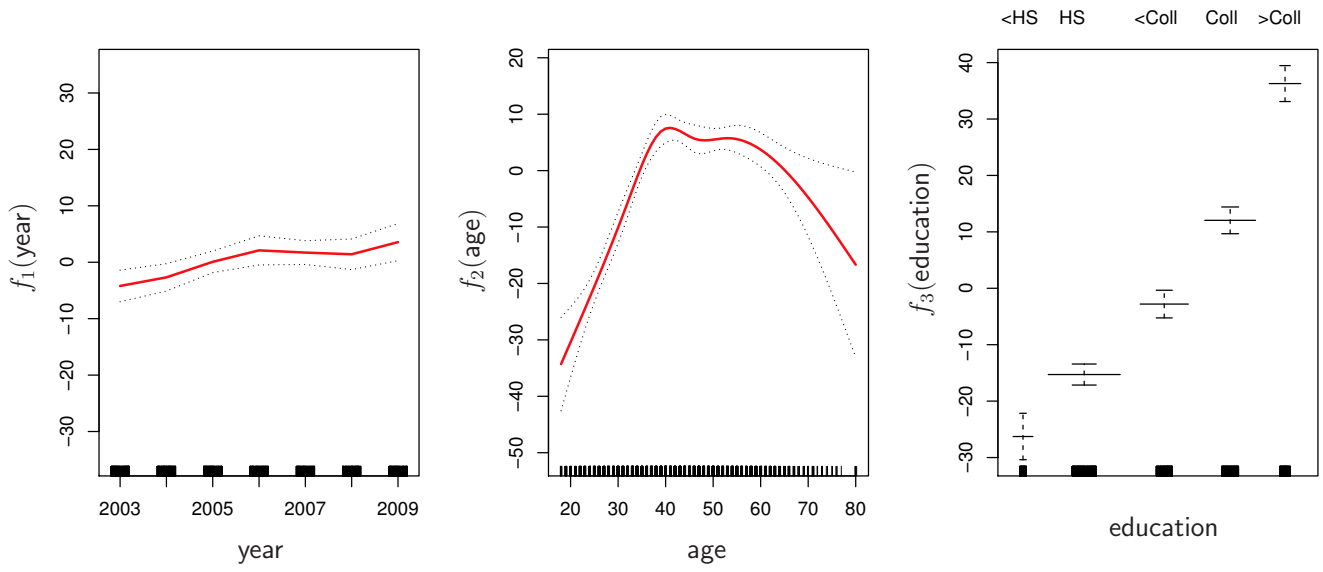


- With a sliding weight function, we fit separate linear fits over the range of X by weighted least squares.
- See text for more details, and *loess*() function in R.

Generalized Additive Models

Allows for flexible nonlinearities in several variables, but retains the additive structure of linear models.

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i.$$

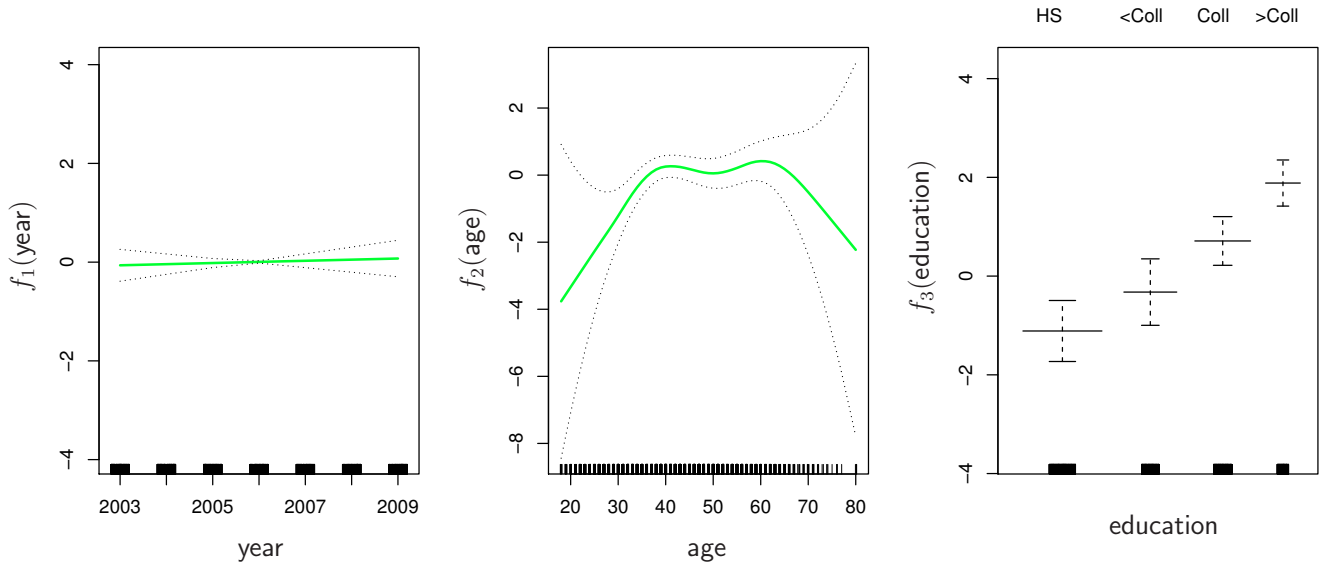


GAM details

- Can fit a GAM simply using, e.g. natural splines:
  $lm(wage \sim ns(year, df = 5) + ns(age, df = 5) + education)$
- Coefficients not that interesting; fitted functions are. The previous plot was produced using *plot.gam*.
- Can mix terms – some linear, some nonlinear – and use *anova*() to compare models.
- Can use smoothing splines or local regression as well:
  $gam(wage \sim s(year, df = 5) + lo(age, span = .5) + education)$
- GAMs are additive, although low-order interactions can be included in a natural way using, e.g. bivariate smoothers or interactions of the form $ns(age, df = 5) : ns(year, df = 5)$.

GAMs for classification

$$log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$



$gam(I(wage > 250) \sim year + s(age, df = 5) + education, family = binomial)$

Tree-based Methods

Tree-based Methods

- Here we describe tree-based methods for regression and classification.
- These involve stratifying or segmenting the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.

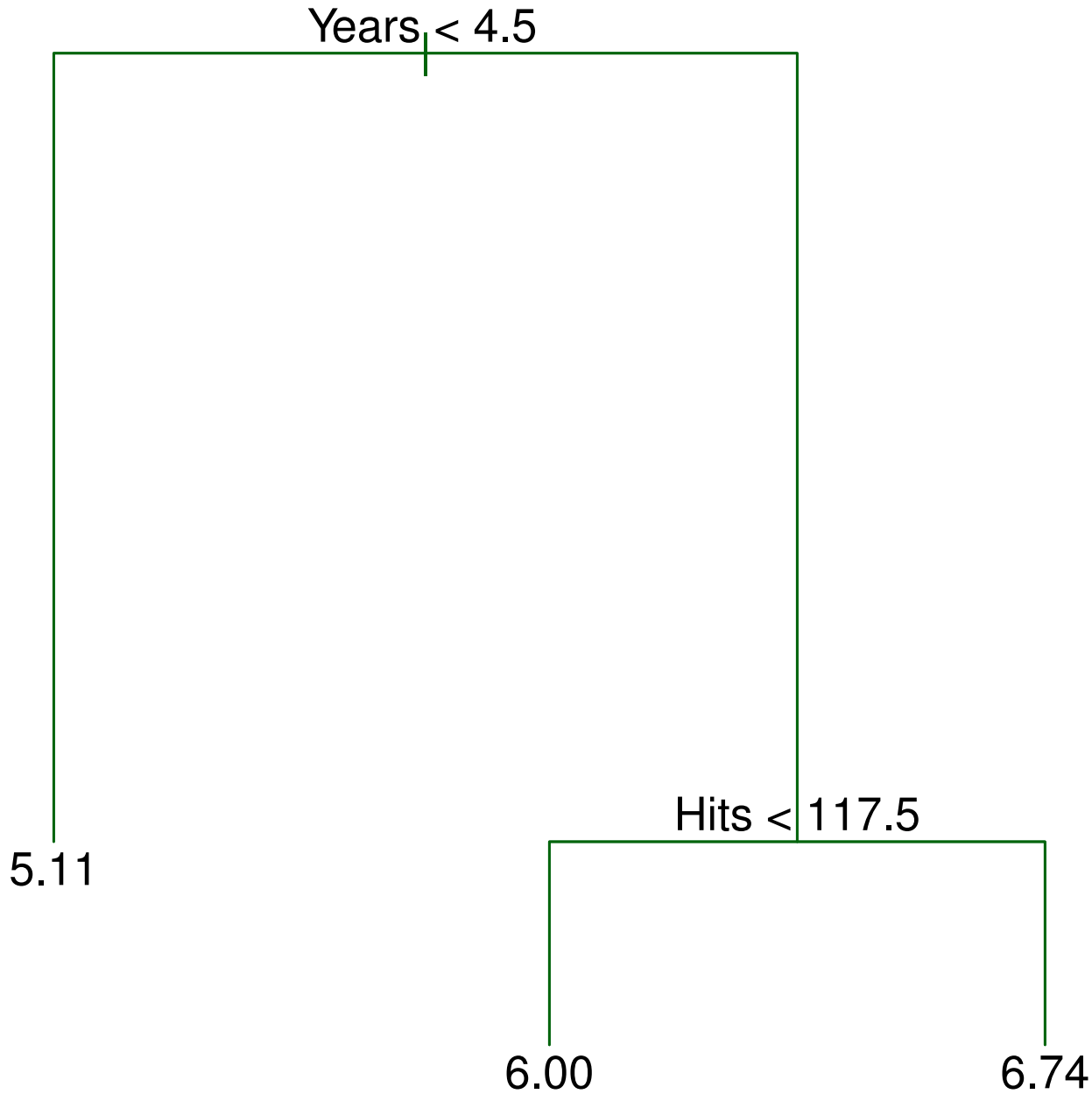
Pros and Cons

- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss bagging, random forests, and boosting. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

The Basics of Decision Trees

- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

Baseball salary data

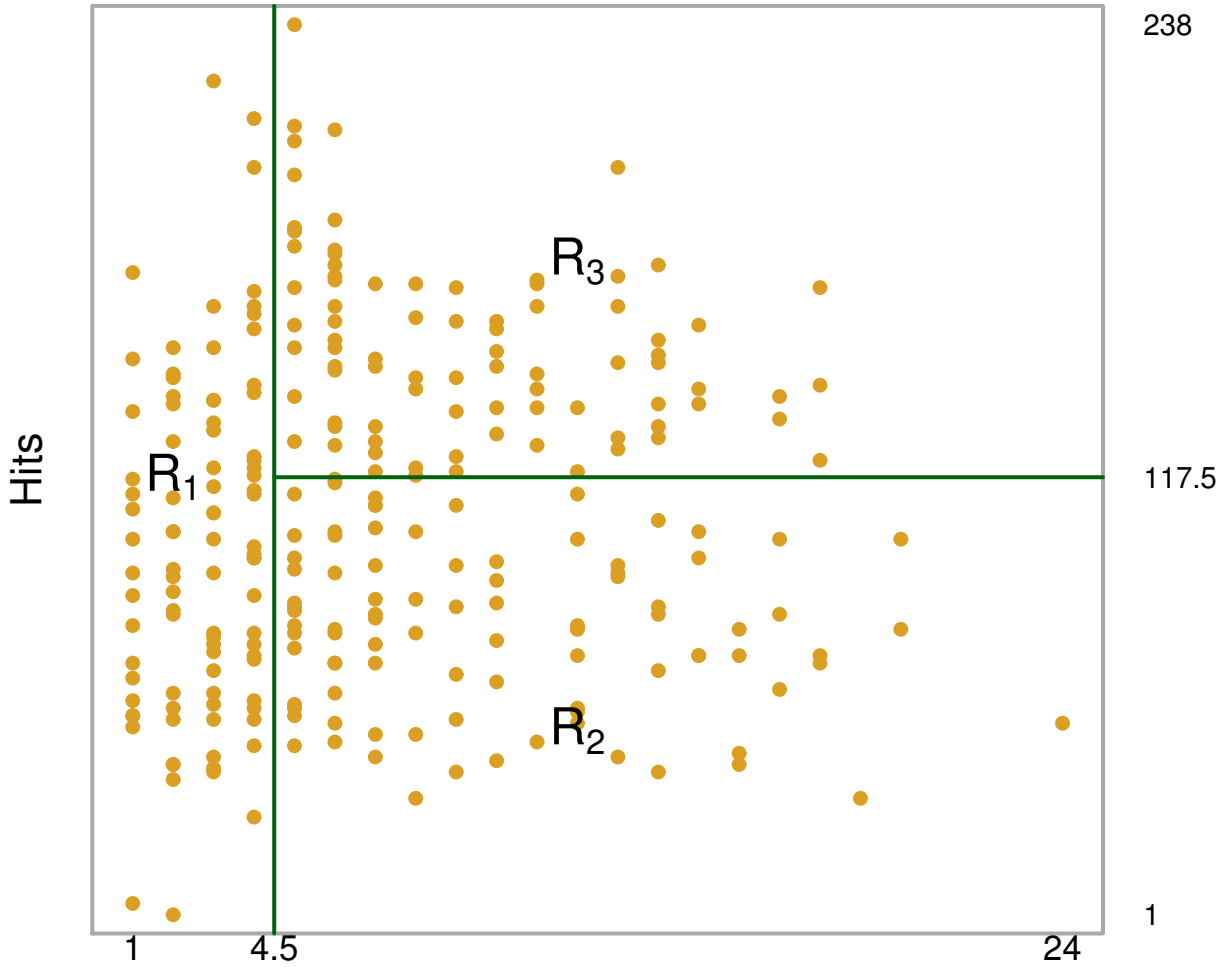


Details of previous figure

- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to $\mathit{Years} < 4.5$, and the right-hand branch corresponds to $\mathit{Years} \geq 4.5$.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Results

- Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X | Years < 4.5\}$, $R_2 = \{X | Years \geq 4.5, Hits < 117.5\}$, and $R_3 = \{X | Years \geq 4.5, Hits \geq 117.5\}$.



Terminology for Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as terminal nodes.
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as internal nodes.
- In the hitters tree, the two internal nodes are indicated by the text $\mathit{Years} < 4.5$ and $\mathit{Hits} < 117.5$.

Interpretation of Results

- *Years* is the most important factor in determining *Salary*, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of *Hits* that he made in the previous year seems to play little role in his *Salary*.
- But among players who have been in the major leagues for five or more years, the number of *Hits* made in the previous year does affect *Salary*, and players who made more *Hits* last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain.

Details of the tree-building process

1. We divide the predictor space – that is, the set of possible values for $X_1, X_2, \ldots, X_p$ – into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

More details of the tree-building process

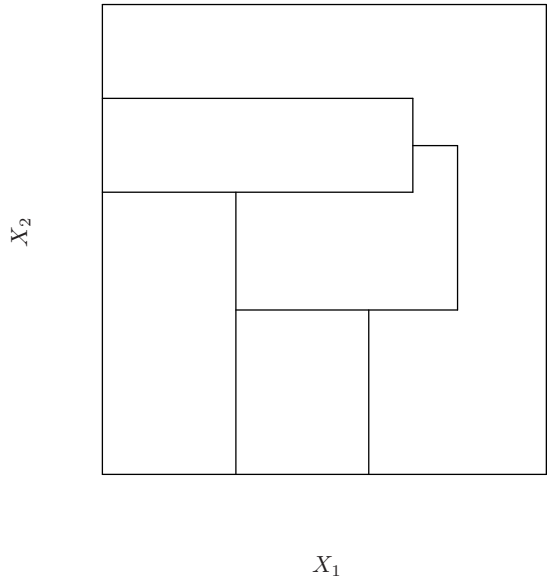
- Unfortunately, it is computationally unfeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

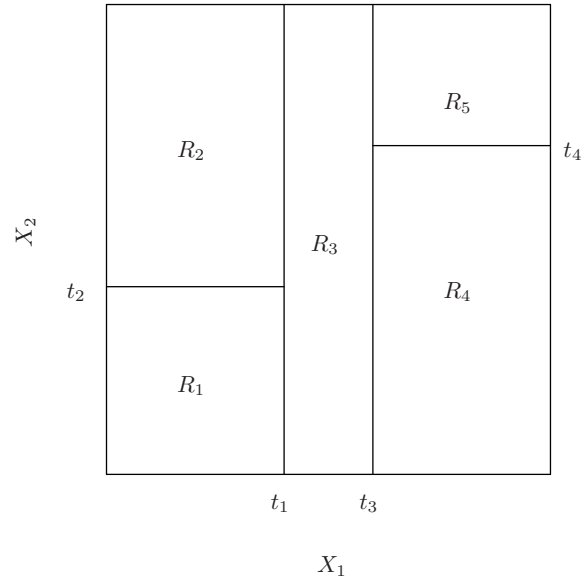
Details – Continued

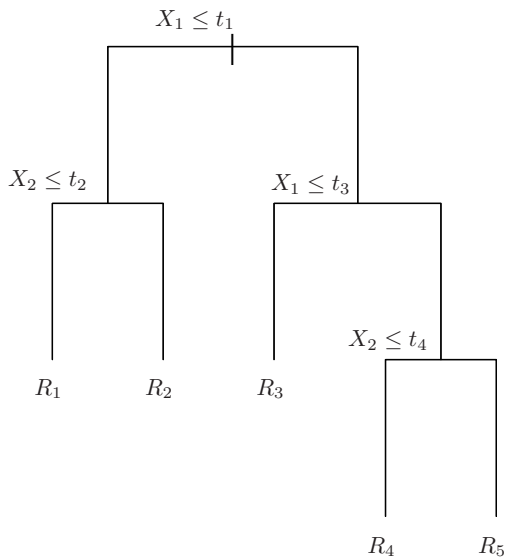
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

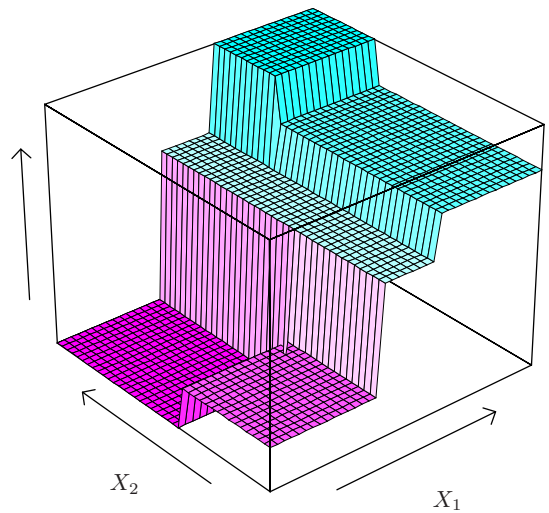
Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.


X_2
X_1


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_2
X_1


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

Details of previous figure

- Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.
- Top Right: The output of recursive binary splitting on a two-dimensional example.
- Bottom Left: A tree corresponding to the partition in the top right panel.
- Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split – that is, a split that leads to a large reduction in RSS later on.

Pruning a tree – continued

- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree.
- Cost complexity pruning – also known as weakest link pruning – is used to do this.
- we consider a sequence of trees indexed by a non-negative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

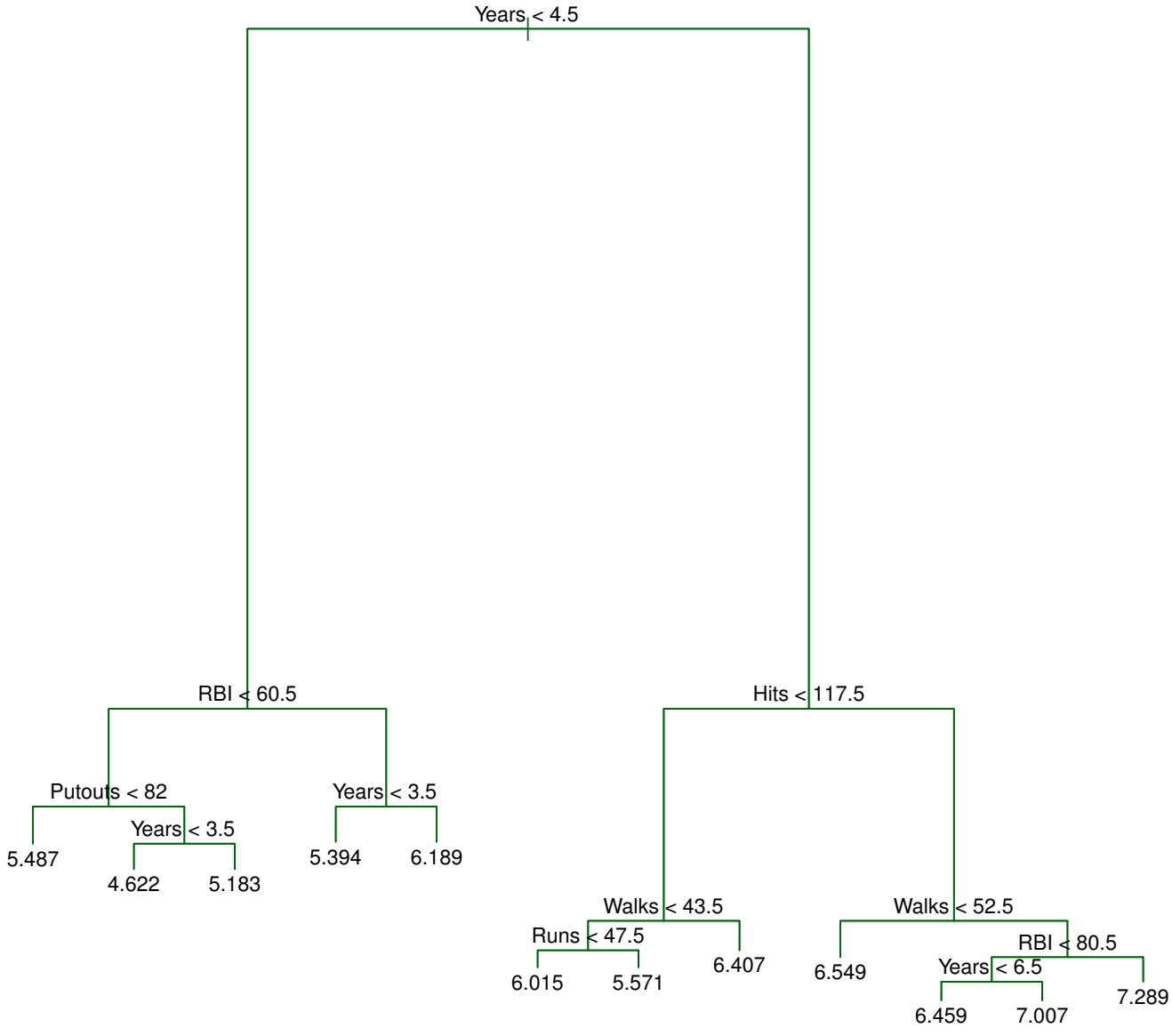
Summary: tree algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

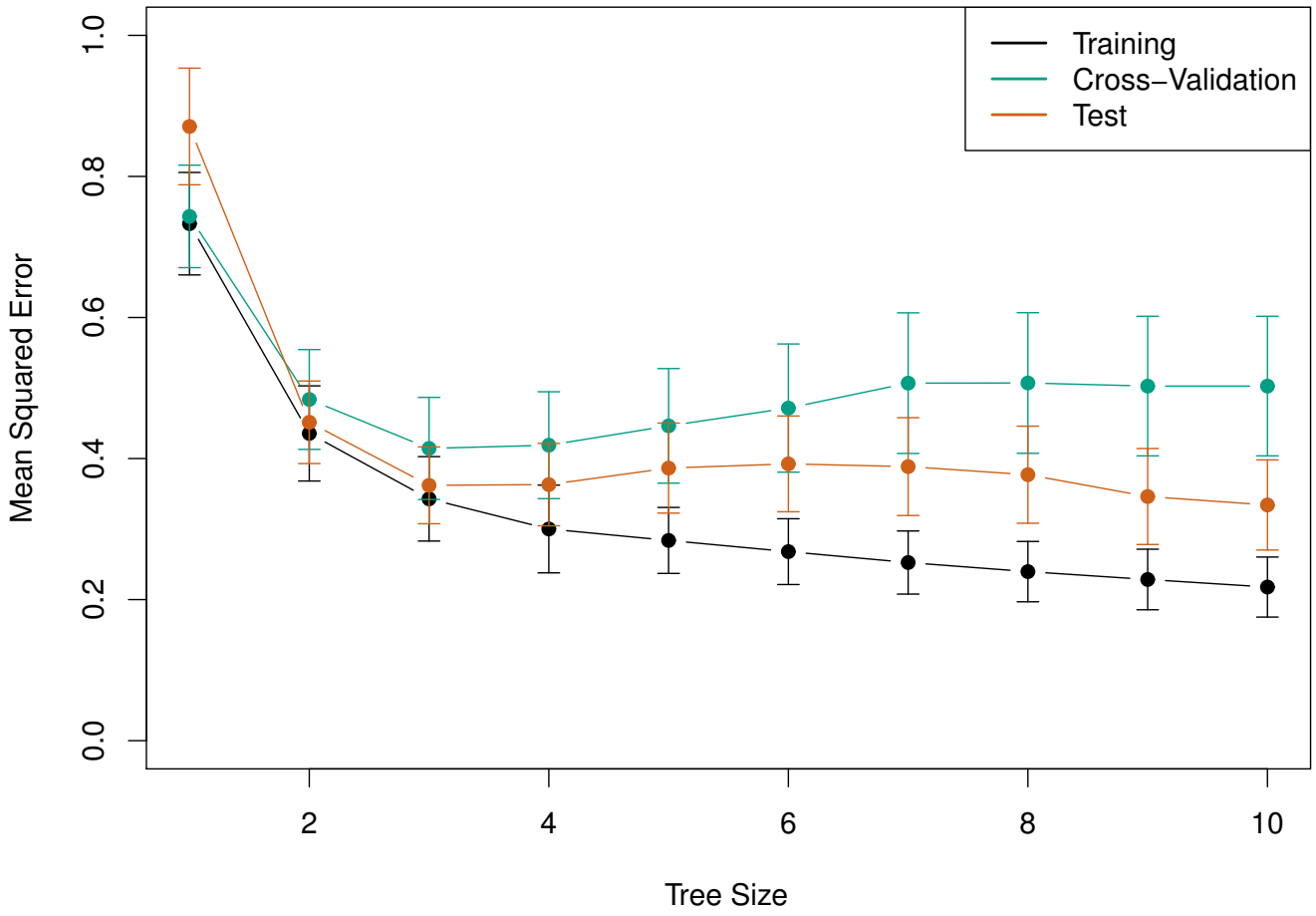
Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



Baseball example continued



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits.
- A natural alternative to RSS is the classification error rate. This is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \underbrace{max}_{k}(\hat{p}_{mk}).$$

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

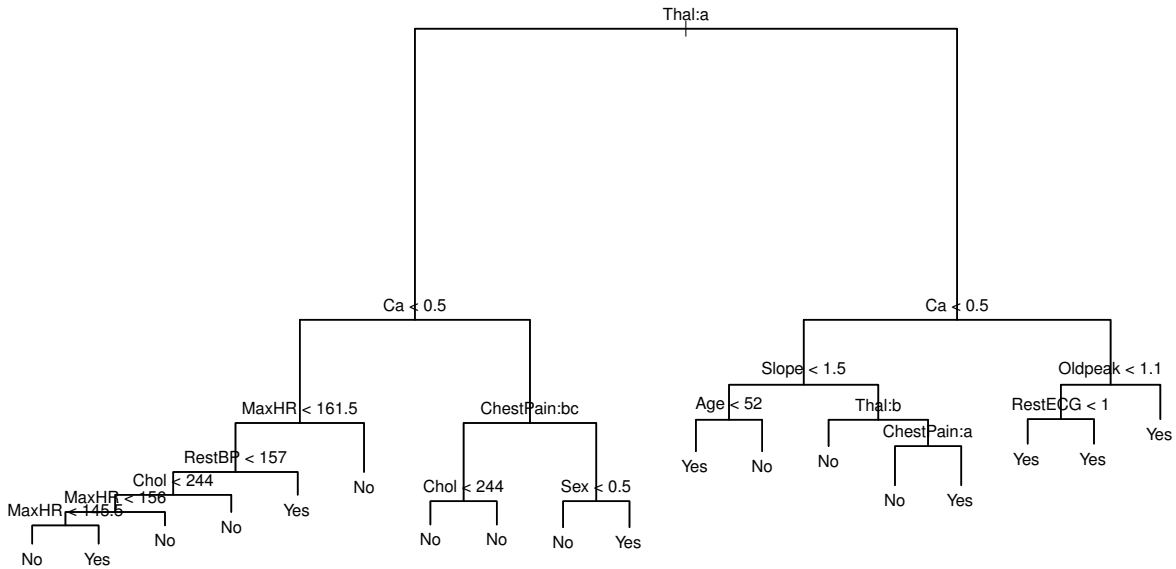
- The Gini index is defined by

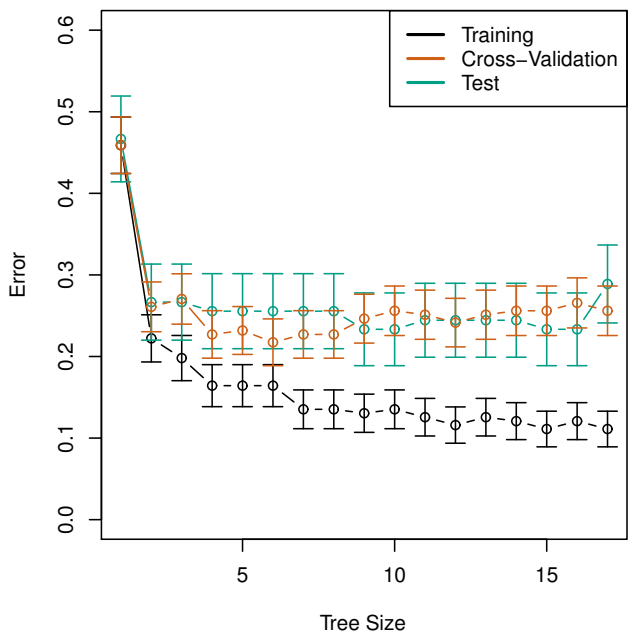
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

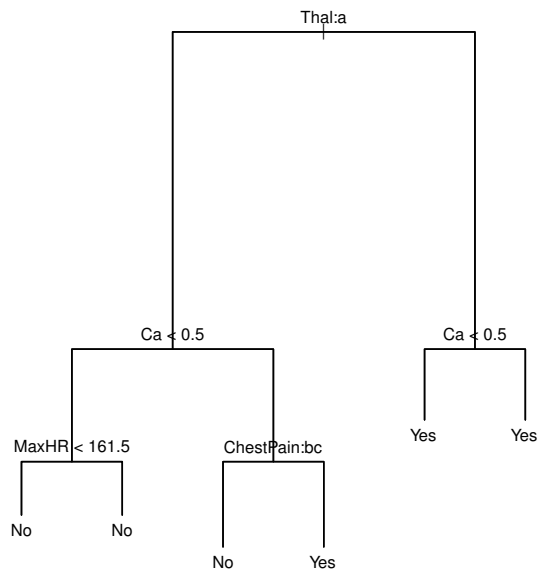
  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node purity – a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is cross-entropy, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk}.$$

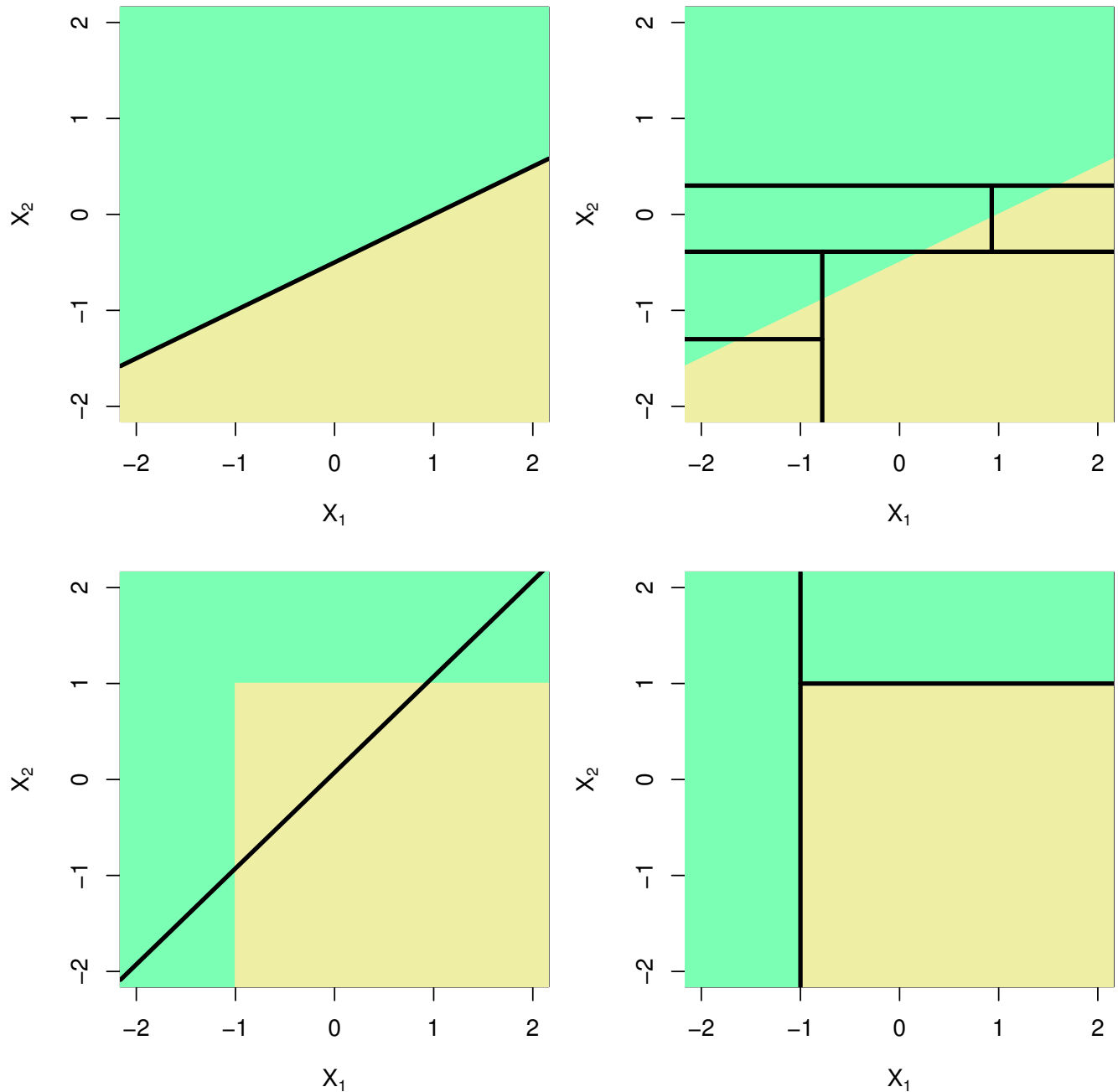
- It turns out that the Gini index and the cross-entropy are very similar numerically.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
Age < 52
Thal:b
RestECG < 1
Yes
RestBP < 157
Chol < 244
MaxHR < 156
MaxHR < 145.5
Chol < 244
Sex < 0.5
ChestPain:a
No
No
No
No
No
No
No
No
No
Yes
Yes
No
No
Yes
Yes
Yes
Yes
No
Yes


Training
Cross−Validation
Test
Error
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15
Tree Size


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Yes
Yes
No
No
No
Yes

Trees Versus Linear Models



Advantages and Disadvantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression.
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches we've seen previously.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Bagging

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

- This is called bagging.

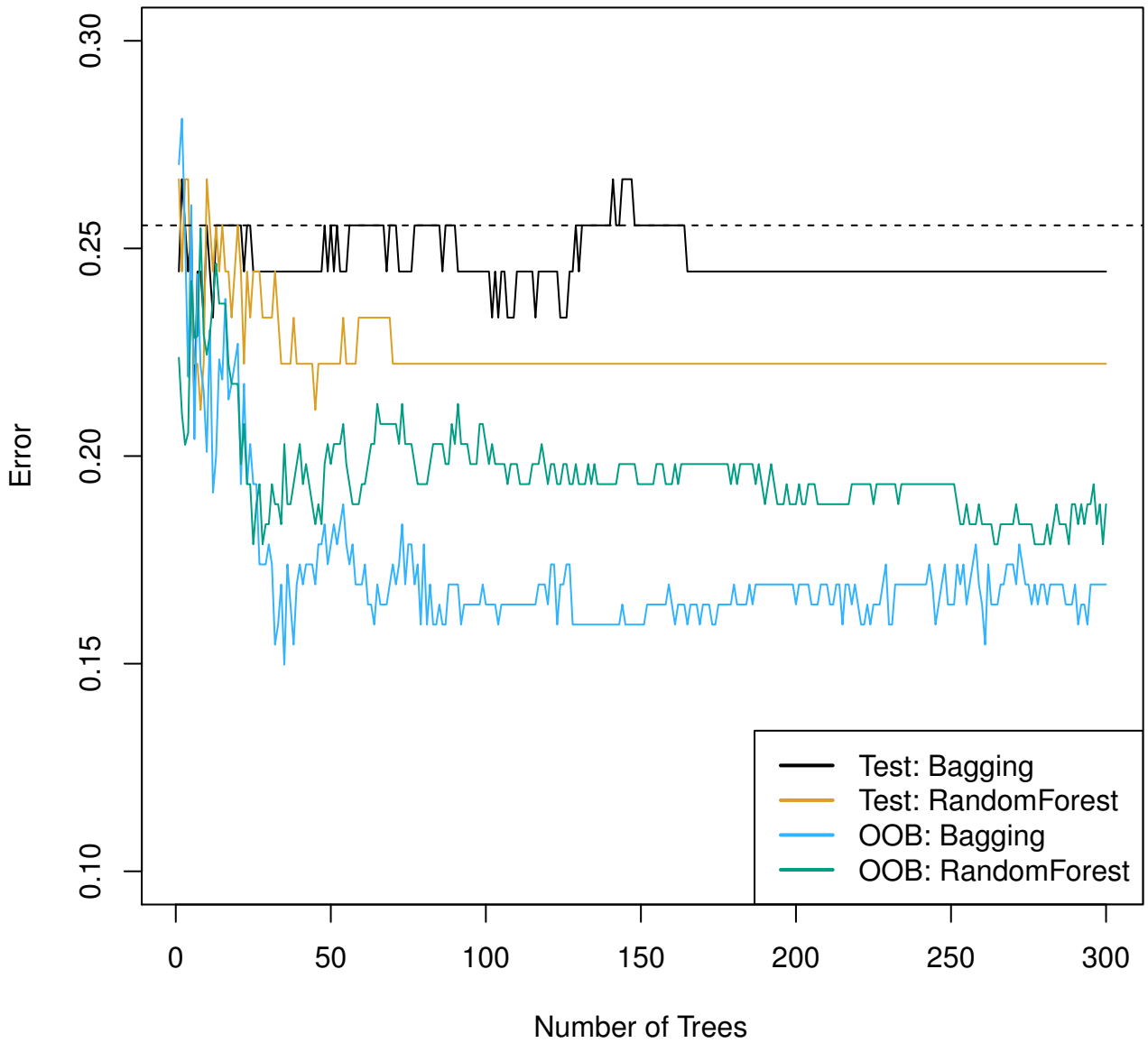
Bagging classification trees

- The above prescription applied to regression trees.
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average.
- This estimate is essentially the LOOCV error for bagging, if B is large.

Bagging the heart data



Details of previous figure

Bagging and random forest results for the Heart data.

- The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used.
- Random forests were applied with $m = \sqrt{p}$.
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the OOB error, which in this case is considerably lower.

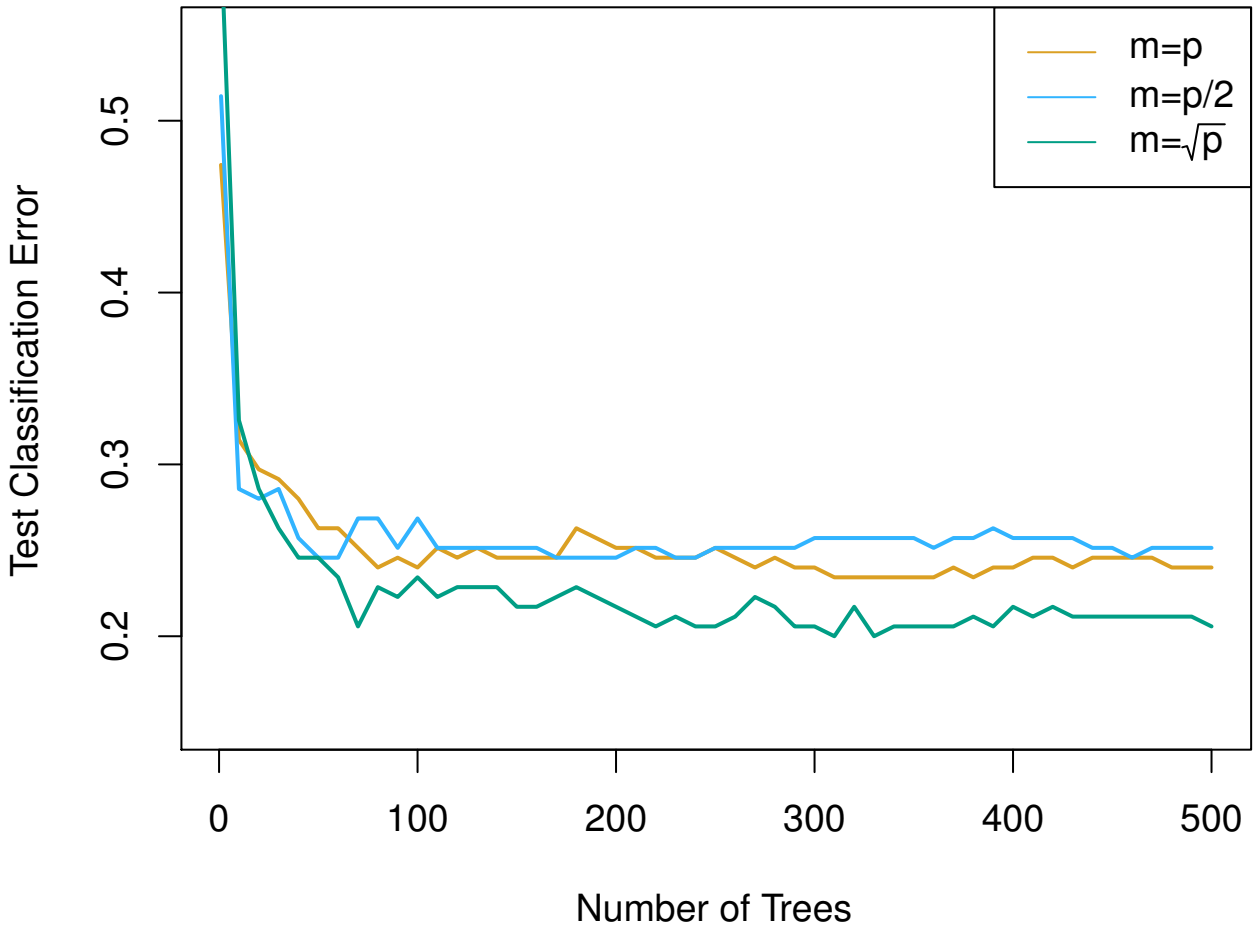
Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ – that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

Example: gene expression data

- Random forests applied to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- Random forests are used to predict cancer type based on the 500 genes that have the largest variance in the training set.
- Observations are randomly divided into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Results: gene expression data



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Boosting algorithm for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   - 2.1 Fit a tree $\hat{f}^b$ with d splits $(d + 1)$ terminal notes to the training data (X, r).
   - 2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

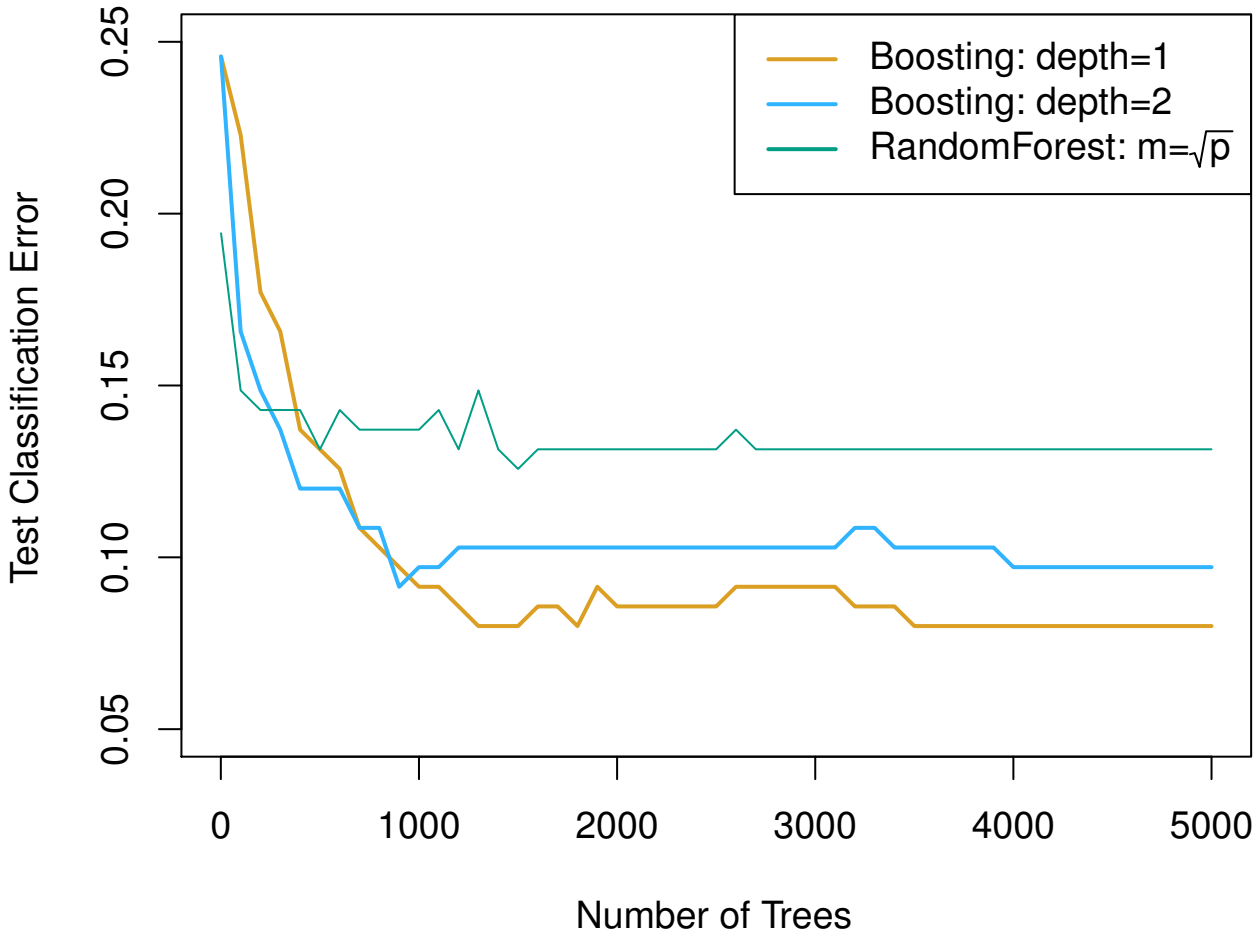
What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here.
- You can find more details in Elements of Statistical Learning, chapter 10.
- The R package *gbm* (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



Details of previous figure

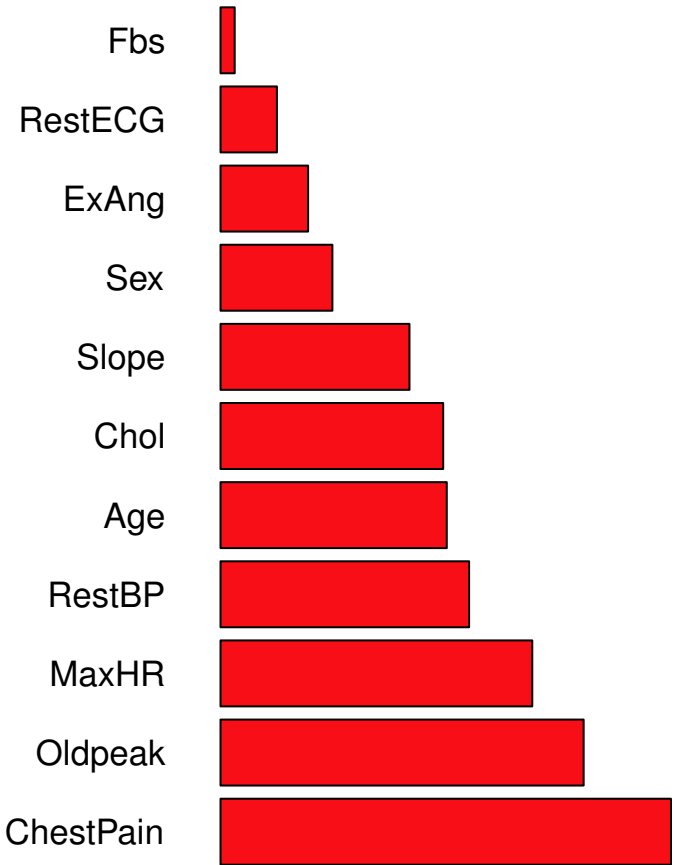
- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict cancer versus normal.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Tuning parameters for boosting

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Summary

- Decision trees are simple and interpretable models for regression and classification.
- However they are often not competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods – random forests and boosting – are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Application example

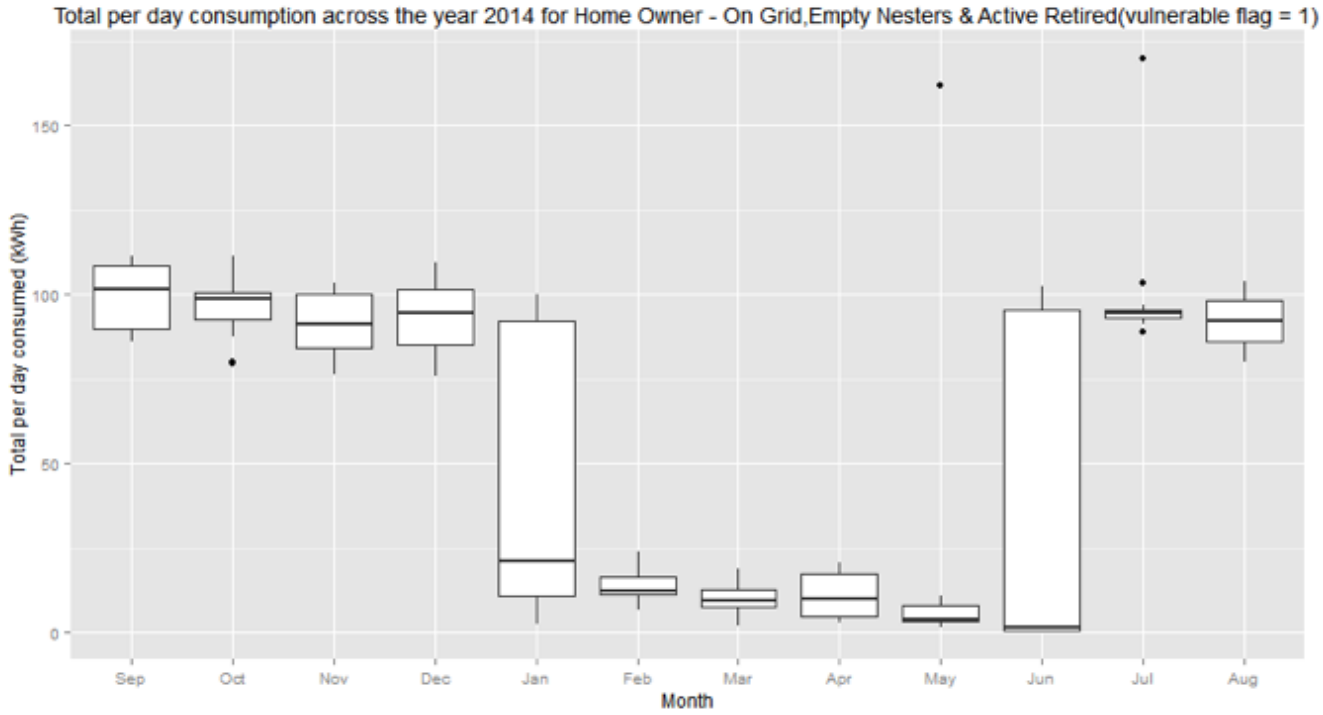
Problem

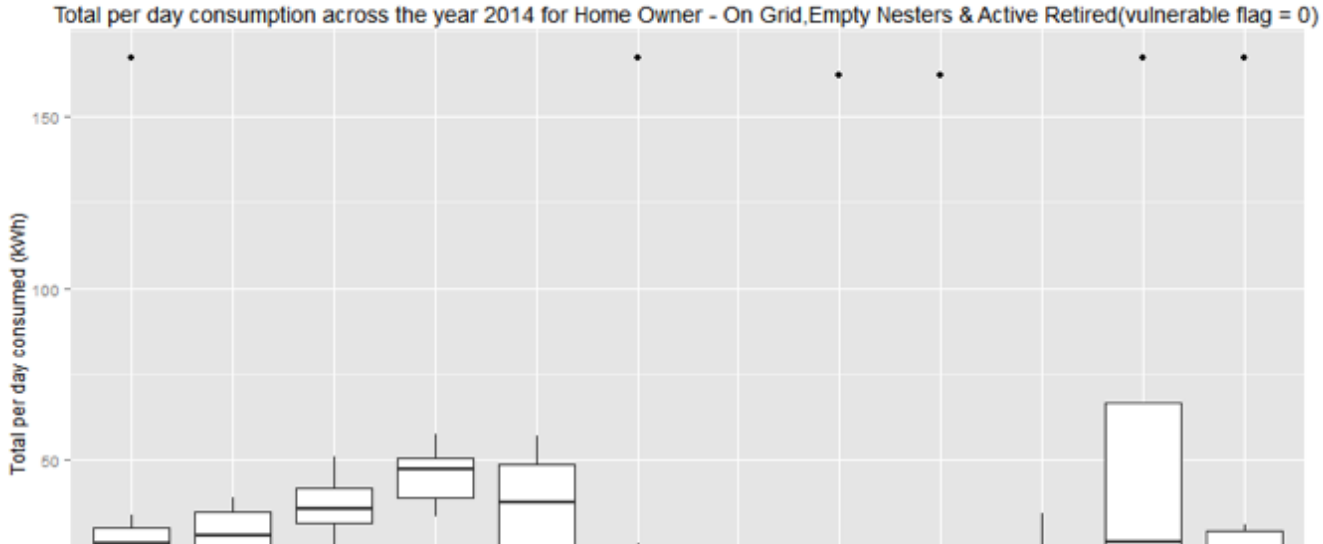
- Energy companies and identification of fuel poor.
- Energy Company Obligations (ECO) and the Green Deal.
- Conventional methods expensive and not agile.
- Consumption data available from smart meters.
- "Predicting Energy Customer Vulnerability from Consumption Behavior" (with Anastasia Ushakova, UCL Consumer Data Research Centre).

Data

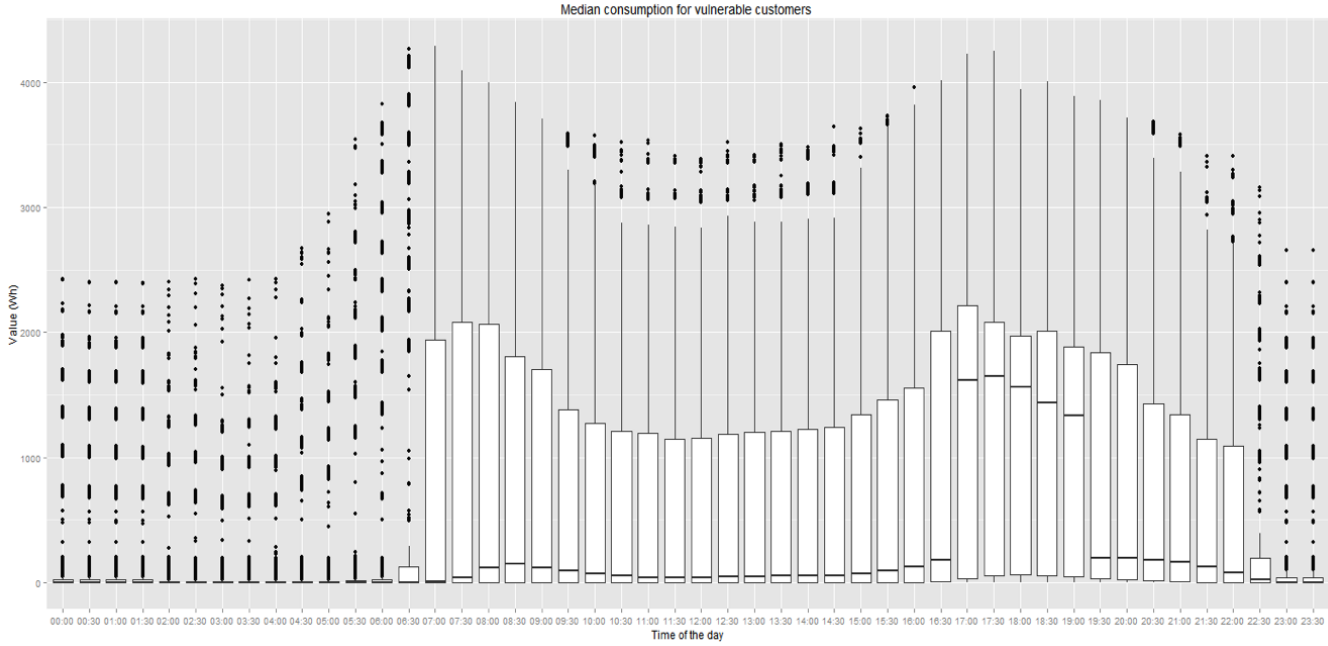
Data	N (smart meters)	N (customers)	N (days)	N (daily readings)	N (total observations)
One month sample	1,919	49,429	28	48	2,372,592
Overall dataset	1,919	693,940	390	48	33,309,120

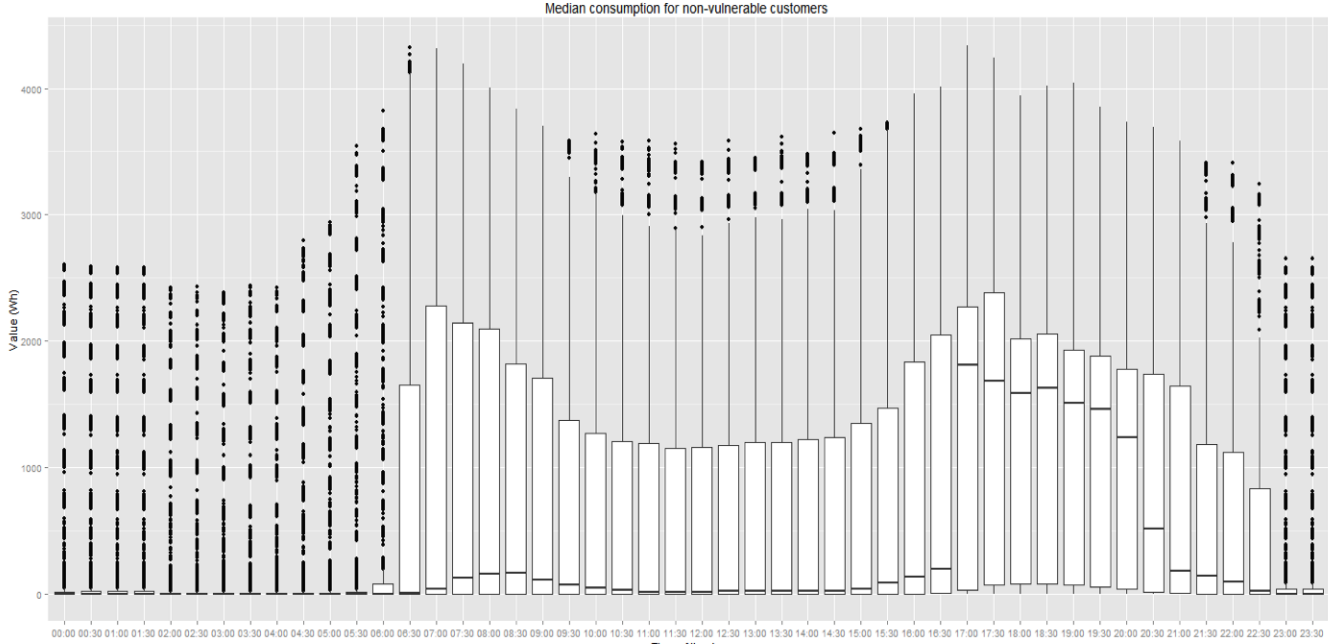
Retired consumers





Aggregate

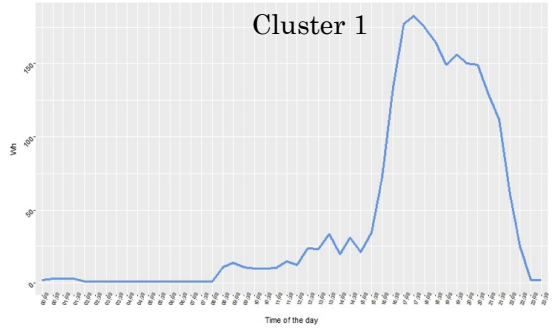

Median consumption for vulnerable customers
Value (Wh)
Time of the day

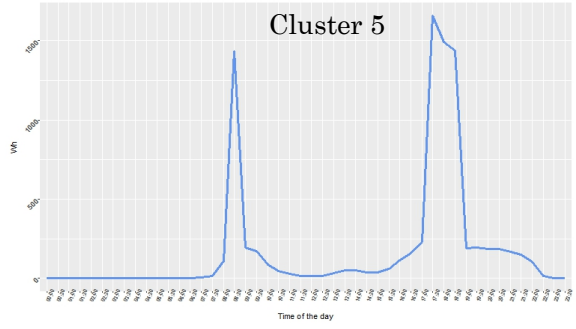

Median consumption for non-vulnerable customers
Value (Wh)

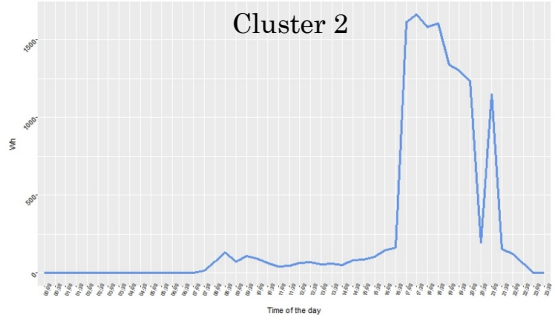
Cluster analysis

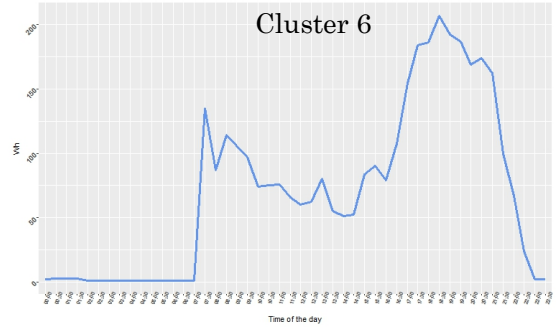
Type	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	cluster7	cluster8
Vulnerable	15.23%	15.18%	6.24 %	20.60%	12.25%	7.06%	8.28%	15.14%
Non-vulnerable	6.43%	22.23%	21.91%	16.81%	15.31%	17.23%	*	*

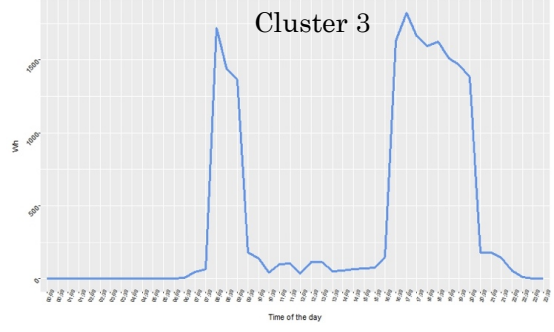
Clustering patterns for vulnerable customers

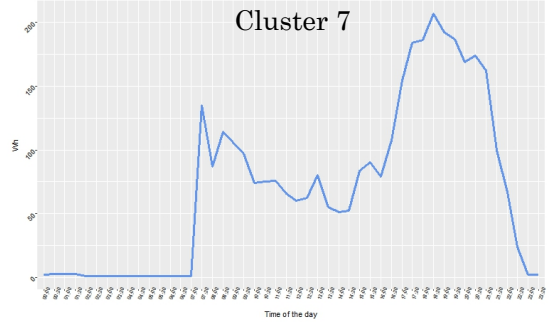




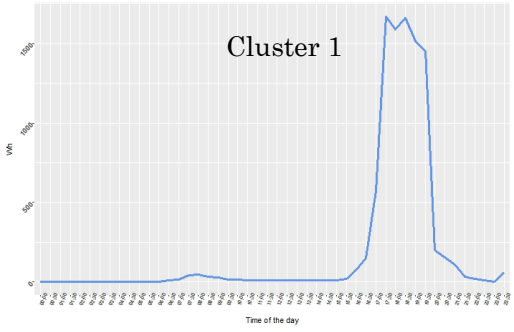


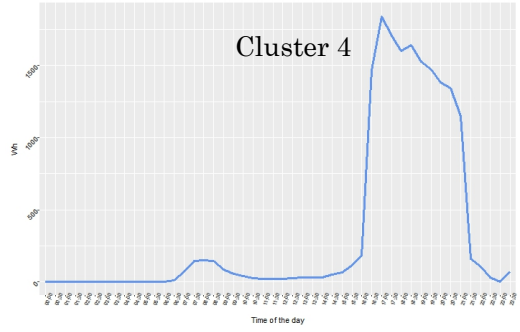


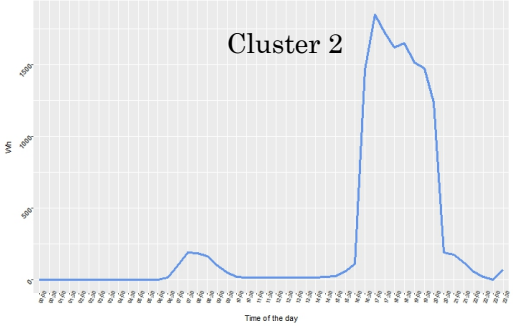


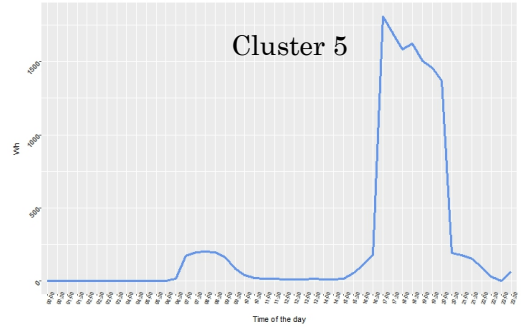


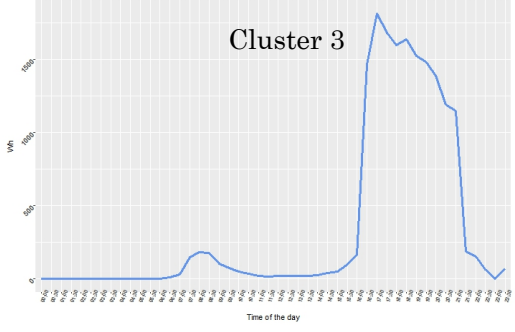
Clustering patterns for nonvulnerable customers

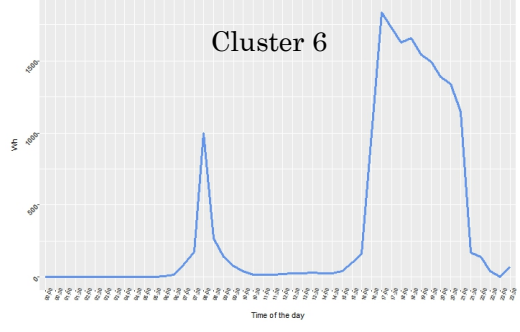








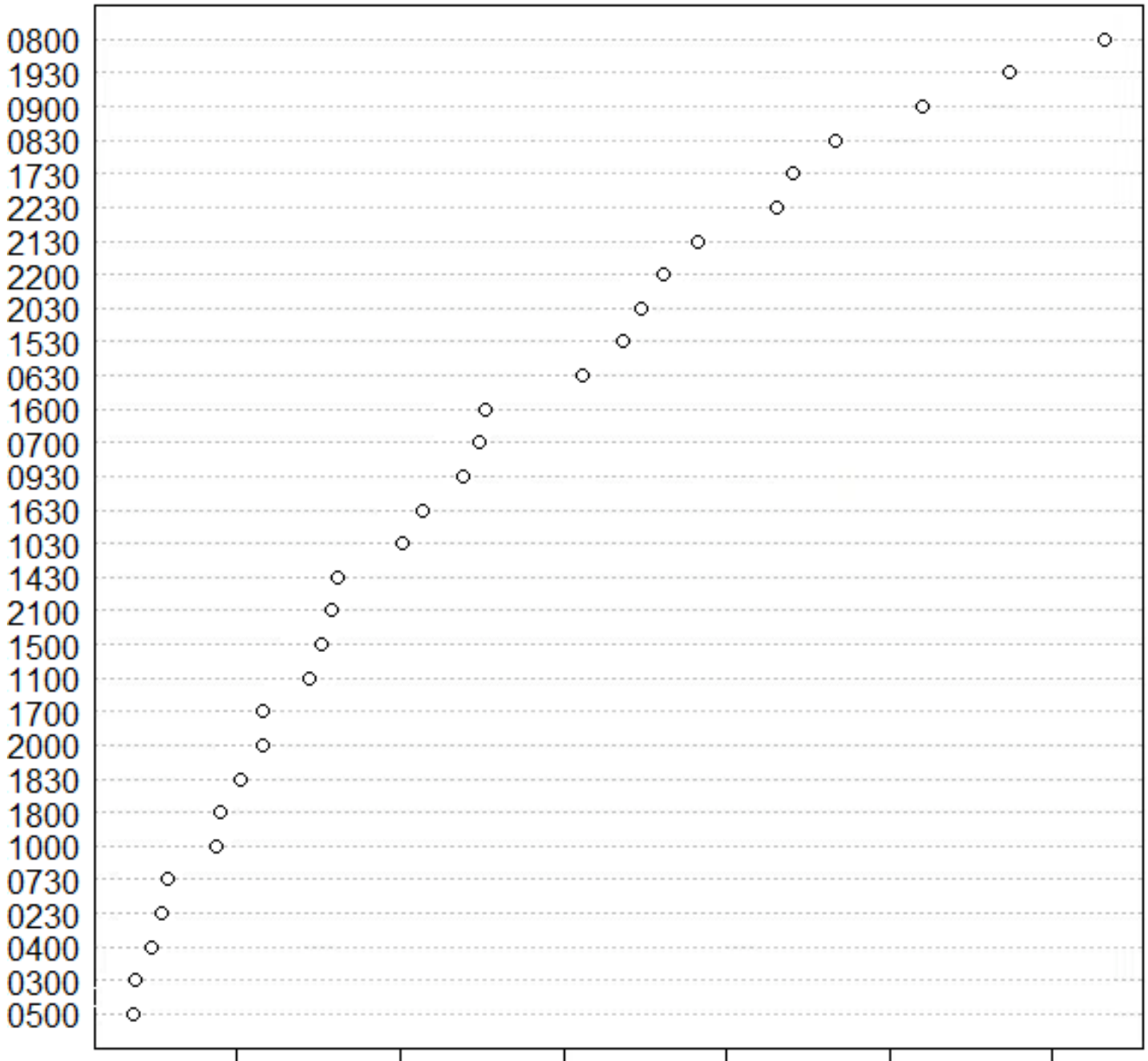


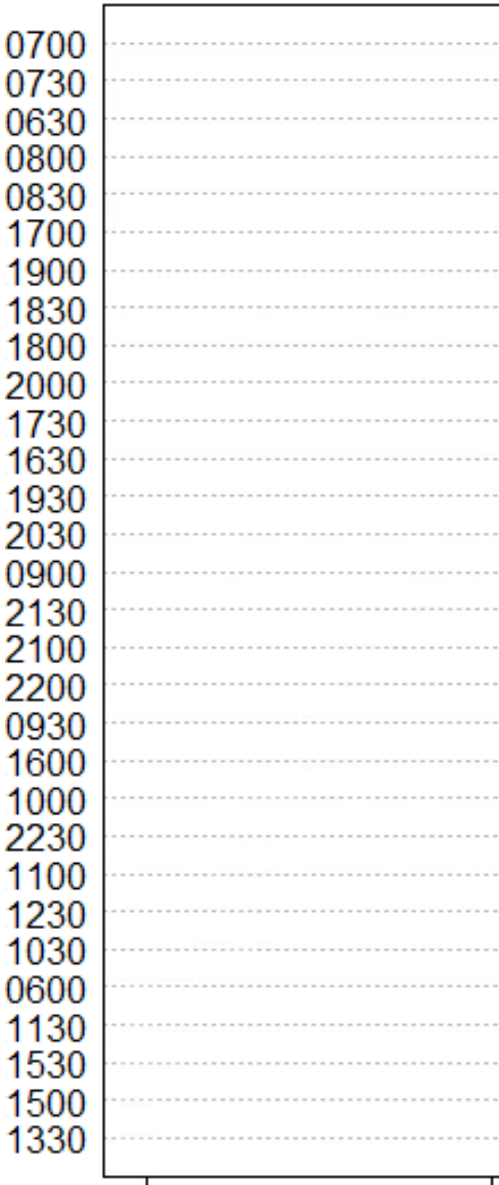


Prediction model

Model	Accuracy	Recall	Precision	F-score
Neural Network	60.11%	0.64	0.66	0.65
SVM (radial kernel)	76.2%	0.81	0.99	0.89
Naive Bayes	56.0%	0.81	0.85	0.83
Random Forest	**94.6%**	**0.81**	**0.79**	**0.80**

Key predictors and validity





Conclusions

- Can predict fuel poverty from consumption behavior.
- Shortening the feedback loops for energy company decision makers (and regulators).
- Improve analysis with linkage of building stock, socio-demographic, and weather data.